UNIVERSITY OF THESSALY
SCHOOL OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Implementation of a mechanism for joint central frequency, bandwidth and station association in WiFi networks

Diploma Thesis

Grigorios Iliadis

Supervisor: Athanasios Korakis

Month 2022

UNIVERSITY OF THESSALY
SCHOOL OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Implementation of a mechanism for joint central frequency, bandwidth and station association in WiFi networks

Diploma Thesis

Grigorios Iliadis

Supervisor: Athanasios Korakis

Month 2022

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Υλοποίηση μηχανισμού για την από κοινού επιλογή κεντρικής συχνότητας, εύρους και συσχετισμού σταθμών σε ασύρματα δίκτυα WiFi

Διπλωματική Εργασία

Γρηγόριος Ηλιάδης

Επιβλέπων/πουσα: Αθανάσιος Κοράκης

Μήνας 2022

Approved by the Examination Committee:

Supervisor **Athanasios Korakis**
Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Antonios Argyriou**
Associate professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Dimitrios Bargiotas**
Professor, Department of Electrical and Computer Engineering, University of Thessaly

Acknowledgements

I want to thank my parents, Dionysios and Zoi, as well as my friends, for their inspiration and support during my academic career.

I want to thank Professor Athanasios Korakis, my supervisor, for giving me the chance to interact with a research setting that involves cutting-edge studies.

I also want to express my gratitude to Senior Researcher and Postdoctoral Ilias Syrigos, who was constantly available to me, guiding and nourishing me with insightful counsel and aiding me in overcoming the roadblocks I experienced in my projects.

I also want to express my sincere gratitude to Professor Dimitrios Bargiotas and Associate Professor Antonios Argyriou as members on the committee that reviewed my thesis.

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

«Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I also declare that the results of the work have not been used to obtain another degree. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism».

The declarant

Grigorios Iliadis

Diploma Thesis

Implementation of a mechanism for joint central frequency, bandwidth and station association in WiFi networks

Grigorios Iliadis

Abstract

With the growing demand for high-speed internet, the need for efficient and effective WiFi networks has also increased. One of the critical aspects of a WiFi network is its ability to handle multiple users and provide seamless handover between Access Points. In this context, the implementation of a mechanism for joint selection of central frequency, bandwidth, and station association in WiFi networks can bring significant benefits. This mechanism aims to enable seamless handover of stations proactively and provide better load balancing, thereby maximizing the throughput of Stations. Especially we use the 802.11ac WiFi protocol which is one of the latest versions with significantly increased data transfer rates. This protocol operates on the 5GHz frequency band, which provides more bandwidth than the 2.4GHz band used by previous WiFi protocols. Also we introduce Deep Q Learning (DQN) algorithm in order to achieve load balancing, which is a critical aspect of efficient WiFi network design. With the current WiFi network design, load balancing is done manually, leading to inefficient use of resources and decreased throughput. The above mechanism uses machine learning algorithm to analyze network traffic and allocate resources dynamically, such as proper channel selection in 5GHz band. The machine learning algorithm is trained with data collected from metrics of the wireless card drivers in Access Points. To validate and evaluate our work in a real environment, we utilize the NITOS Testbed to deploy a simple WiFi topology of two Access Points (AP) and four Stations (STA) and we examine different association scenarios between them. This architecture uses a back haul node (BH) in which machine learning and a python3 application server is deployed in order achieve communication between access points and back haul node for data collection. This gives the advantage of remote control of the network. Experiments happened in real-world environments and with the use of simulation tool. Our results denote that stations can be efficiently distributed among available channels, while the near-future traffic can be effectively forecast using our Machine Learning

approach.

Keywords:

WiFi, Deep Q Learning (DQN), 802.11ac, load balancing (LB), seamless handover

Διπλωματική Εργασία

Υλοποίηση μηχανισμού για την από κοινού επιλογή κεντρικής συχνότητας, εύρους και συσχετισμού σταθμών σε ασύρματα δίκτυα WiFi

Γρηγόριος Ηλιάδης

Περίληψη

Με την αυξανόμενη ζήτηση για υψηλής ταχύτητας ίντερνετ, η ανάγκη για αποτελεσματικά και αποδοτικά δίκτυα WiFi έχει αυξηθεί επίσης. Ένα από τα κρίσιμα στοιχεία ενός δικτύου WiFi είναι η δυνατότητά του να χειρίζεται πολλαπλούς χρήστες και να παρέχει απρόσκοπτη μετάβαση ανάμεσα στα σημεία πρόσβασης. Σε αυτό το πλαίσιο, η εφαρμογή ενός μηχανισμού για την από κοινού επιλογή κεντρικής συχνότητας, εύρους ζώνης και σύνδεσης σταθμών σε δίκτυα WiFi μπορεί να φέρει σημαντικά οφέλη. Ο συγκεκριμένος μηχανισμός αποσκοπεί στην επιτροπή απρόσκοπτης μετάβασης των σταθμών και στη βελτίωση της ισορροπίας φόρτου, επιτυγχάνοντας έτσι τη μέγιστη επίδοση των σταθμών. Ειδικότερα, χρησιμοποιούμε το πρωτόκολλο WiFi 802.11ac το οποίο είναι μία από τις πιο πρόσφατες εκδόσεις με σημαντικά αυξημένες ταχύτητες μεταφοράς δεδομένων. Αυτό το πρωτόκολλο λειτουργεί στη ζώνη συχνοτήτων των 5GHz, που παρέχει περισσότερο εύρος ζώνης από τη ζώνη συχνοτήτων των 2,4GHz που χρησιμοποιούνται από προηγούμενα πρωτόκολλα WiFi. Επίσης, εισάγουμε τον αλγόριθμο Deep Q Learning (DQN) για να επιτύχουμε την ισορροπία φόρτου, που αποτελεί ένα κρίσιμο στοιχείο του αποδοτικού σχεδιασμού των δικτύων WiFi. Με τον τρέχοντα σχεδιασμό δικτύου WiFi, η ισορροπία φόρτου γίνεται χειροκίνητα, οδηγώντας σε ανεπαρκή χρήση πόρων και μειωμένες ταχυτήτες. Ο παραπάνω μηχανισμός χρησιμοποιεί αλγόριθμο μηχανικής μάθησης για την ανάλυση της κίνησης στο δίκτυο και τη δυναμική κατανομή πόρων, όπως η κατάλληλη επιλογή καναλιού στη ζώνη των 5GHz. Ο αλγόριθμος μηχανικής μάθησης εκπαιδεύεται με δεδομένα που συλλέγονται από μετρικές των drivers των ασυρμάτων καρτών δικτύου στα Access Points (AP). Για να επικυρώσουμε και να αξιολογήσουμε τη δουλειά μας σε ένα πραγματικό περιβάλλον, χρησιμοποιούμε το NITOS Testbed για να αναπτύξουμε μια απλή τοπολογία WiFi με δύο Access Points (AP) και τέσσερεις Stations (STA) και εξετάζουμε διαφορετικά σενάρια συνδεσιμότητας μεταξύ

τους. Αυτή η αρχιτεκτονική χρησιμοποιεί έναν back haul (BH) κόμβο στον οποίο είναι εγκατεστημένος ο αλγόριθμος μηχανικής μάθησης και ένας application server σε python3, προκειμένου να επιτευχθεί η επικοινωνία μεταξύ των Access Points και του back haul κόμβου για τη συλλογή δεδομένων. Αυτό προσφέρει το πλεονέκτημα του απομακρυσμένου ελέγχου του δικτύου. Τα πειράματα πραγματοποιήθηκαν σε πραγματικά περιβάλλοντα αλλα και με τη χρήση εργαλείου προσομοίωσης. Τα αποτελέσματά μας δείχνουν ότι οι stations μπορούν να διανεμηθούν αποτελεσματικά μεταξύ των διαθέσιμων καναλιών, ενώ η κίνηση στο μέλλον μπορεί να προβλεφθεί αποτελεσματικά χρησιμοποιώντας την προσέγγιση της μηχανικής μάθησης.

Λέξεις-κλειδιά:

WiFi, Deep Q Learning (DQN), 802.11ac, ισσοροπία φορτίου, απρόσκοπτη μετακίνηση

Table of contents

List of figures

Abbreviations

AES-CCMP	AES-Counter Mode CBC-MAC Protocol
AP	Access Point
BH	Back Haul
BSS	Basic Service Set
BSSID	Basic Service Set Identifier
CERTH	Centre for Research and development Hellas (CERTH)
CSA	Channel Switch Announcement
CSM	Client Steering Mechanism
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DQN	Deep Q Network
FT	Fast BSS Transition
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
LB	Load Balancing
ML	Machine Learning
MPD	Markov Desision Tree
NITlab	Network Implementation Testbed Laboratory
NITOS	Network Implementation using Open-Source software
OMF	Control and Management Framework
QoS	Quality of Service
RL	Reinforcement Learning
RSSI	Received Signal Strength Indicator
TCP	Transmission Control Protocol
MIMO	Multiple-Input Multiple-Output

MU MIMO	Multiple-User Multiple-Input Multiple-Output
SSID	Service Set Identifier
SSL	Secure Sockets Layer
STA	Station
SU MIMO	Single-User Multiple-Input Multiple-Output
TLS	Transport Layer Security
UDP	User Datagram Protocol
WiFi	Wireless Fidelity
WPA2	Wi-Fi Protected Access 2

Chapter 1

Introduction

1.1 Motivation

The demand for fast and reliable access to the internet through wireless networks from a constantly growing number of users and devices highlight them as a crucial part of our daily lives. At the same time new challenges arise facing problems related to the seamless and efficient operation of wireless networks.

Congestion is one major problem that affects WiFi networks because of the increasing number of devices connected to the internet. The consequences are saturation of the available bandwidth, latency issues, slow speeds and unstable connections with periodically disconnections. As solutions in congestion problems are preferred the upgrading of network infrastructure, limitation in number of users as long with the application of Quality of Service (QoS) politics in the network. Approaches like this aim to prioritize traffic, or implement bandwidth throttling to limit the amount of bandwidth that can be used by individual devices or applications.

The limited range of WiFi networks can also be a problem especially in large buildings or outdoor areas. Several factors determine the range of networks such as transmission power of Access Points , antenna design and surrounding obstacles. Network administrators often use spectrum analyzing tools to examine the coverage area and equip the network with range extenders or resort to mesh network topology with multiple Access Points for seamless coverage over a larger area.

At last but not least WiFi networks operate transmissions in unlicensed bands where a variety of other devices can also operate especially in houses and public areas, such as cord-

less phones, Bluetooth devices, and microwave oven. This is a major drawback in network performance because it can not be controlled by the network designers. They can only take measures for minimizing interference like appropriate channel selection using spectrum analyzing equipment to detect the source of interference and more advanced antennas less vulnerable to interference.

1.2 Thesis subject

The ability of a network to handle efficiently multiple users and to provide seamless handover between Access Points especially in mobility scenarios is critical aspect. Given that fact we implement a mechanism for joint central frequency, bandwidth, and station association in WiFi networks which focuses on maximizing throughput for stations with load balancing techniques and proactively handover of stations. Consequently we will examine the benefits of a mechanism like this which is based on mobilization of Machine Learning for optimization load balancing and resource allocation problems.

A significant advantage of joint mechanism is that allows the efficient use of the available bandwidth and as a result decreases the interference and maximizes the performance of stations. It ensures that Access Points operate on different frequencies and adjusts channel bandwidth and central frequency dynamically in contrast with the current design of the WiFi networks where Access Points often operate in same frequencies and use fixed channels to transmit.

One other aspect of the above mechanism, especially useful in large public spaces such as airports, malls, train stations etc, is seamless handover of station between Access Points without momentarily disconnection of stations. In contrary with applicable mechanism for Access Point selection which is based on signal strength and by extension distance from Access Point our implementation allows to associate stations proactively. Taking into advantage the global knowledge of the network, the centralized approach it can be useful in load balancing and allows proactive re-associations which are critical especially in movement scenarios.

Finally centralized network design makes it possible to introduce Machine Learning (ML) in our mechanism for detecting patterns and taking actions such as station association. Load balancing in current WiFi networks is difficult because of the frequent changes in network state and the external interference. Machine learning algorithms are powerful and capable

with proper modeling of network to analyze traffic and allocate resources dynamically in never seen before ways. It is important to mention that we train our algorithm at first offline in simulation environment and then online in a real WiFi topology with multiple Access Points and stations.

1.2.1 Contribution

In this section we present numerically the actions and methodologies of this thesis which are solving the problems we promised in the previous union:

1. Brief presentation of the previous WiFi protocols compared to the 802.11ac
2. Examine the features of the 802.11ac which is used in our implementation.
3. Analyze the capabilities of 802.11r protocol which gives the ability to stations to switch between access points without the need for a complete re-association process.
4. Implementation of the client steering mechanism in order to achieve seamless re-association of station between our network's access points.
5. Introduction of the Reinforcement Learning algorithm and examination of different application approaches.
6. Examination of the data collection process from the Ath10k driver metrics and synthesis process of the data set used for algorithm training.
7. Implementation of the Deep Q Learning algorithm.
8. Off-line algorithm training with simulation data.
9. Nitos testbed wireless WiFi network topology construction and DQN algorithm online training and experimentation with different association scenarios.
10. Evaluation of DQN algorithm

1.3 Content Organization of Thesis

This Thesis is divided into seven chapters. In chapter 2 we present the evolution of WiFi protocols up to the 802.11ac protocol which is used in our implementation. In chapter 3 the

default station association process in WiFi is shown and is compared with the client steering mechanism proposed for seamless handover between Access Points. Chapter 4 consists of an introduction to Reinforcement Learning (RL) and presentation of Deep Q Learning Algorithm (DQN) which is proposed in this thesis. All the experimental tools used to exploit our implementation are shown in chapter 5. In chapter 6 we present the experimental topology exploited in NITOS Tesbed and we evaluate the DQN algorithm performance. Finally, an overall evaluation of the proposed mechanism and future work prospects are discussed in chapter 7.

Chapter 2

WiFi Protocols

2.1 WiFi protocols previous to 802.11ac

WiFi, or wireless fidelity, is a popular wireless communication technology used for connecting devices to the internet or local networks without the need for cables. The first WiFi protocol was released in 1997, and since then, there have been several updates and improvements to the technology, each with its own set of characteristics.

In 1999 two WiFi protocols were released. The first protocol that used 5GHz band was the 802.11a, which achieved higher speeds relative to previous technologies approximately up to 54 Mbps and aimed to performance improvement for multimedia applications, such as video streaming and gaming. Even though it had advantages especially because it operated on a less congested frequency band, which meant that it was less susceptible to interference from other devices, it could not penetrate through walls and other obstacles. That lead to shorter range compared to other technologies that operated in 2.4GHZ band. Then the 802.11b protocol was introduced which used the 2.4GHz band. This protocol despite achieving lower speed rates, approximately at 11Mbps, had better performance in terms of penetration through obstacles and transmission range. In addition with compatibility with existing hardware became the most commercial WiFi protocol for homes, businesses, and public spaces.

They followed two more protocol releases. In year 2003 802.11g was released which used the 2.4GHz band and despite it achieved speeds same with 802.11a protocol was only compatible with 802.11b. Among the advantages compared with 802.11a was larger range and penetration capabilities through walls and natural obstacles. However on the downsides it was still vulnerable in interference from other devices that operate in the same frequency

band. Then in year 2009 a significantly improved protocol compared to the previous was released. This was 802.11n and achieved transmission rates up to 600 Mbps, i.e ten times more than the previous 802.11a and 802.11g, with the use of both frequency bands of 2.4GHz and 5GHz. Alongside in this protocol MIMO (multiple-input and multiple-output) technology was introduced, which allowed for the use of multiple antennas to improve data throughput and range. Also it is improved in terms of error correction and interference resistance from other devices. Finally it was compatible with the previous WiFi protocols and so it was applicable in devices operating with 802.11a-b-g which had as result the wide use of it and significant upgrade of existing wireless networks without large cost.

IEEE Standard	802.11a	802.11b	802.11g	802.11n	802.11ac	802.11ax
Year Released	1999	1999	2003	2009	2014	2019
Frequency	5Ghz	2.4GHz	2.4GHz	2.4Ghz & 5GHz	2.4Ghz & 5GHz	2.4Ghz & 5GHz
Maximum Data Rate	54Mbps	11Mbps	54Mbps	600Mbps	1.3Gbps	10-12Gbps

Figure 2.1: WiFi Standards

2.2 802.11ac WiFi Features

The most recent and advanced WiFI protocol is 802.11ac, which offers new features and improvements compared to previous protocol versions. In this section, we will explore the features of 802.11.ac WiFi and compare it with its predecessors. Below we collocate some of the most important 802.11.ac WiFi Features:

1. Increased Data Transfer Rates: Achievement increased data transfer rates, aspect that makes it ideal for applications that require high-speed internet such as video streaming and online gaming, is due to operation in 5GHz frequency band. This band provides more bandwidth than the 2.4GHz band with channels that can be 20 MHz, 40 MHz, 80 MHz, and 160 MHz wide.

2. Multi-User MIMO (MU-MIMO): The use of this technology makes it possible for multiple devices to simultaneous receive data from a single access point, negating the

need for devices to shift while accessing the network. Also allows synchronous data transmission in multiple devices which reduces the congestion of the network and improves overall throughput. Areas with high traffic (e.g. airports, shopping malls and universities) can benefit the most from this mode.

3. Beamforming: This technology allows access points to direct wireless signals to specific devices, improving signal strength and reducing interference. Beamforming is particularly useful in large spaces where access points need to provide coverage to a large number of devices.

4. Improved Security: 802.11.ac WiFi also provides improved security features over previous protocols. This protocol supports the latest encryption standards, including WPA2 and AES-CCMP, which provide enhanced security for wireless networks. Additionally, 802.11.ac WiFi allows for the use of certificate-based authentication, which ensures that only authorized devices can connect to the network.

5. Backward Compatibility: One of the significant advantages of 802.11.ac WiFi is its backward compatibility with previous WiFi protocols. Devices that use older protocols, such as 802.11a, b, g, and n, can still connect to an 802.11.ac network. However, the data transfer rates for these devices will be limited to the capabilities of their respective protocols.

2.2.1 Comparison with Previous WiFi Protocols

The 802.11ac protocol first of all with MU-MIMO technology achieves to restrict networks congestion and improve overall throughput giving the ability in multiple devices to transmit simultaneously. It is an innovative feature, in comparison with previous protocols, which support only Single-User MIMO (SU-MIMO), i.e. only one device at a time to transmit data. In addition, in regard to Beamforming, the 802.11ac WiFi protocol provides more complex Beamforming capabilities, advanced in comparison with 802.11n in which Beamforming was also available, because it allows in access points to direct signal wireless in specific devices. Finally, excels in data transfer rates due to the use of the 5 GHz frequency band, which provides more bandwidth than the 2.4 GHz band used in previous protocols. So, 802.11ac can approach theoretically as a maximum data transfer rate 6.77 Gbps, exceeding

by far the 600 Mbps, which are listed as the maximum theoretical data transfer rate for the 802.11n.

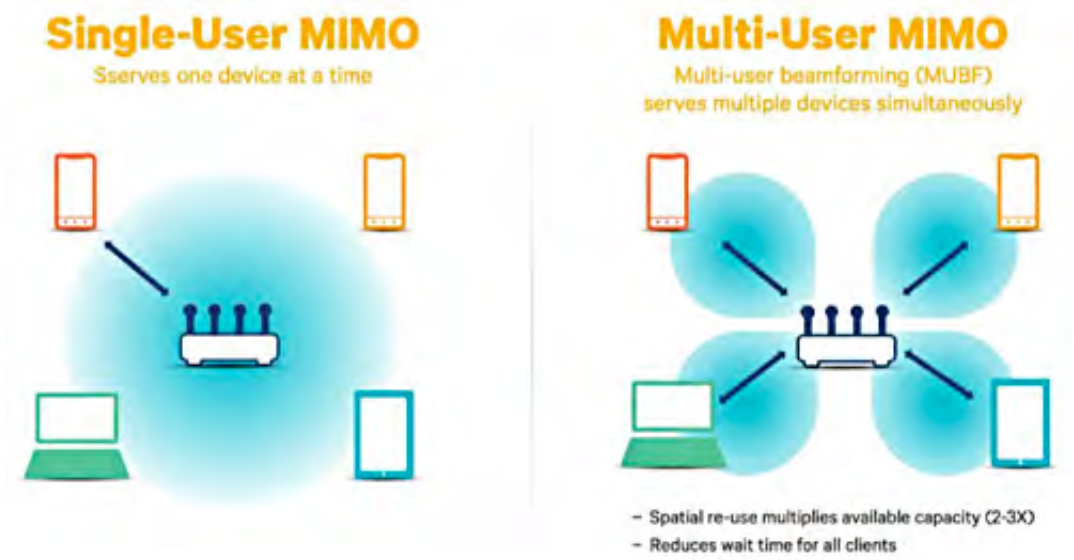

Figure 2.2: SU MIMO vs MU MIMO

2.3 IEEE 802.11r standard

The IEEE 802.11r standard, also known as Fast BSS Transition (FT), is an evolution of the previous protocols in terms of providing enhanced roaming capabilities and is useful in wireless application because guarantees in users uninterrupted network connectivity. It was developed to overcome the problems of slow access and loss of connectivity in WiFi networks during roaming procedure of mobile devices between access points, due to the requirement of a full re-association process with the new access point. In other words when a device moves towards a new access point traditionally must resubmitted the to the complete time-consuming authentication and security handshake process.

Indeed with the 802.11r standard the goal of more efficient and faster roaming process was achieved as devices now can switch between access points without undergoing full re-association process. Fast BSS Transition (FT) is the technique that allows in a device to quickly switch to a new access point when moving out of the range of the associated access point without performing a complete re-association process. This is possible with pre-authentication of the device with the new access point while it is still connected to the current access point.

It is also important to mention that 802.11r contributed significantly to improvement of the overall performance of the wireless network. Provides seamless and secure roaming experience for users minimizing the time it takes for a device to roam between access points. As a consequence it helps to minimize disruptions in network services and reduces network congestion by enabling devices to quickly connect to an available access point, reducing the load on congested access points.

2.3.1 Relation between 802.11ac and 802.11r protocols

While the two protocols serve different purposes, they can work together to improve overall wireless network performance. For example, when a mobile device with IEEE 802.11ac capabilities roams between access points that support IEEE 802.11r, it can benefit from the seamless and secure handover provided by 802.11r, resulting in uninterrupted network connectivity and a better user experience.

In addition, the higher data transfer rates and improved performance provided by IEEE 802.11ac can help to support more simultaneous connections, reducing network congestion and enabling more devices to connect to the network. This can be especially important in high-density WiFi environments such as airports, stadiums, and other public venues.

Overall, in our implementation we take advantage of 802.11r capabilities in order to implement a client (station) steering mechanism. This mechanism ensures seamless re-association of stations between network's access points. We will examine analytically the implementation of client steering mechanism in the following chapter.

Chapter 3

Client steering mechanism

3.1 Default association process in WiFi

In this section we will describe the default association process of a station in case where multiple access points are available, because that is the main characteristic of WiFi network topology examined in this thesis. The factors that determine access point selection is signal strength, quality and load of the available access points, as well as the predefined roaming criteria in order to ensure that the station connects to the access point that provides the best wireless connection.

This process involves the following steps:

1. Scanning: At first the station performs a periodic scan of the available access points, by sending probe request frames containing the SSID of the target network. Then the station listens for probe response frames from access points within range that match the SSID.

2. Signal strength and quality: The station measures the signal strength and quality of the probe response frames received from each access point, because these are important factors in determining available access point with the best wireless connection.

3. Authentication and association: After selecting the access point with the strongest signal and highest quality station sends an authentication request frame. Then if the authentication request is accepted by the access point, the station sends an association request frame to connect to it.

4. Connection quality monitoring: After establishing association the station continuously monitors the quality of the wireless connection. In case where the signal strength and quality of the connection is decreased, the station will probably decided to roam to a different access point with a stronger signal.

5. Roaming decision: There is a set of predefined criteria, such as the signal strength, quality, load of the current access point and neighboring access points, that influence the station decision to roam. In case where the station detects an available access point with potentially better performance, it will trigger re-association process by disconnecting from the current access point and connecting to the new one.

3.1.1 Comparison with client steering mechanism

The above mechanism main characteristic is that is triggered from the station's side. However in our approach we want to proactively trigger this process from a centralized network administration system in order to achieve better load balancing exploiting the holistic supervision of the WiFi network. So, the client steering mechanism we propose in the following section mimics the above process in order to achieve seamless re-association without affecting the station's connection.

3.2 Re-association process

We refer the re-association process as client steering because the centralized network controller, with use of Machine Learning, decides to move stations from one access point (AP) to one other in order to achieve better load balancing and maximize total throughput of stations. Below we will describe the details of the process.

Before diving into the details of the scripts, it's important to have an overview of the re-association mechanism. The purpose of this mechanism is to allow a station (client device) to switch from one access point (AP) to another without disrupting the user's network connection. This is achieved by performing a "soft" handover, where the station first associates with the new acess point and then disassociates from the old one. To accomplish this, the re-association mechanism uses a feature called "Concurrent Service Advertisement" (CSA) in IEEE 802.11 networks. CSA allows the new AP to advertise its presence to the station before the station disassociates from the old access point. This gives the station time to prepare for

the handover and avoid any disruption to the network connection. Be default CSA messages are used when an access point changes central frequency and wants to let stations now the new central frequency in order to keep up with it. However in our implementation the access point sends CSA message without changing central frequency and the new central frequency is of one other access point. In that way the station moves unaware to one other access point. Also, in the default channel switch process the CSA message is broadcast so all stations connected to the AP can receive it and after some beacon time intervals all stations move to the new central frequency where the AP declares in the CSA messages. In our implementation the CSA message is unicast and only the one station we want to steer receives it.

At first we will describe a simplistic topology of tow access points and one station in order to explain the basic steps of the client steering mechanism. The topology consists of tow access points, one station already connected to one of them and one back haul node from whom we send ip traffic towards the station. Both access points, station and back haul node are hosted on different NITOS indoor testbed nodes. We assume that station is initially connected to AP hosted on node063, with mac ”00:03:1d:0d:bc:81”, and the other AP is hosted on node075 with mac ”00:03:1d:0d:bc:9b”. In both access points we set up bridge interfaces, which connect two different interfaces (bridge ports). Bridging two interfaces causes every Ethernet frame that is received on one bridge port to be transmitted to the other port. Thus, the two bridge ports participate in the same Broadcast domain. In our case the bridge is a connection of Ethernet interface of the access point with the Wireless interface. The wireless interface has the same mac for both access points but the Ethernet have different in order the back haul, in our scripts named as node060, can direct the traffic properly among the access points. So the mac addresses mentioned above refer to the Ethernet interface of each access point. From now on we refer to the AP on node 063 as source AP and to the AP on node 075 as target AP, because our goal is to move the station to the second AP without interrupting the connection.

The client steering mechanism contains three bash scripts and one python script. The main bash script which triggers the re-association process is the $client\_steering.sh$ and is executed from the $nitlab3$ server which has direct access to the name space of the nodes described above. From that script the tow additional bash scripts $target.sh$ and $source.sh$ are executed remotely with use of ssh commands. The first is executed in target AP’s terminal and the second in the source AP, where the station is already connected. Below we appose

the three bash scripts.

```
source=$1
target=$2
mac=$3
node3="node060"

if [ $source -eq '36' ]
then
    node1="node063"
    node2="node075"
    targetMAC="00:03:1d:0d:bc:9b"
else
    node1="node075"
    node2="node063"
    targetMAC="00:03:1d:0d:bc:81"
fi

ssh root@$node2 "/bin/bash -s '$mac'" <./target.sh
(ssh root@$node1 "/bin/bash -s '$mac $target'" <./source.sh) &> /dev/null
ssh root@$node3 "arp -s 192.168.2.3 '$targetMAC'"
```

Listing 3.1: client_steering.sh

```
mac=$(echo "$1" | head -n1 | awk '{print $1;}')
echo $mac

python3 CSA.py $1
/root/hostapd-2.6-csa/hostapd/hostapd_cli disassociate $mac
echo hw-restart > /sys/kernel/debug/ieee80211/phy0/ath10k/
    simulate_fw_crash
```

Listing 3.2: source.sh

```
/root/hostapd-2.6-csa/hostapd/hostapd_cli new_sta $1 /root/assoc.txt
```

Listing 3.3: target.sh

The first step of the steering process is to add manually in the target AP, with the use of *hostapd* commands, the association request of the station we want to move so the target AP is aware of the new station's state and capabilities. For that purpose the *hostapd* source code is modified so it takes as argument in the *new_sta* command the mac address of the station

and an association request identical with the one that station does when he enters the network. The $assoc.txt$ in $target.sh$ script is a part of the association request of the station. We make a replica of the association request by capturing one when station enters the network, then we isolate the capabilities of the station and we update them manually in the target AP with $hostapd\_cli$. In that way the process of exchanging probe request and probe response frames is bypassed.

The second step of the client steering process takes place in the source AP. There we execute the $CSA.py$ script in order to inform the station for central frequency switch. The script is written in Python and uses the $scapy$ library to craft and send wireless packets. The purpose of the script is to send a unicast Channel Switch Announcement (CSA) to a particular station (identified by its MAC address) on a specific channel. The CSA packet is constructed with various parameters such as BSSID, SSID, and the new channel to switch to. The CSA packet is then sent using the $scapy$ library's $sendp()$ function. Also we disassociate the station from the source AP. Finally, the script simulates a firmware crash on the WiFi card by writing the string $hw\_restart$ to the $/sys/kernel/debug/ieee80211/phy0/ath10k/simulate\_fw\_crash$ file.

One other thing to mention is that we have to update the back haul node's arp table after the channel switch. The back haul node sends ip traffic, with $iperf3$ connection between it self and station connected in the source AP. At first the back haul node knows that station is in source AP so the traffic is send with destination the mac of the bridge (Ethernet) interface of source AP. After client steering the station changes to target AP and we must update the arp table of the back haul node. The mac which is related with the station must now be associated with the bridge (Ethernet) interface of target AP and the traffic to be send through target AP in order to reach the station.

3.2.1 Summary

Overall, in the default channel switch process the AP moves to new central frequency and then the stations move in the new frequency expecting to found the AP there. So we exploit that and we send a similar CSA message declaring that after some beacon interval the AP will move in the new central frequency. But in our implementation the message is unicast ,i.e. we send it in one specific station, and the AP does not move in one other frequency. We already know that in the new central frequency exists an AP with the same mac in which we have

previously added the state of the station we want to move, i.e. all the variables that declare the capabilities of the station, the mac address of the station etc. **The wireless interface of all access points have the same mac**, so the station can not recognize the switch and sends the packets with same way as before. **In this way we make the handover transparent.**

Below there is a schematic depiction of the client steering process.

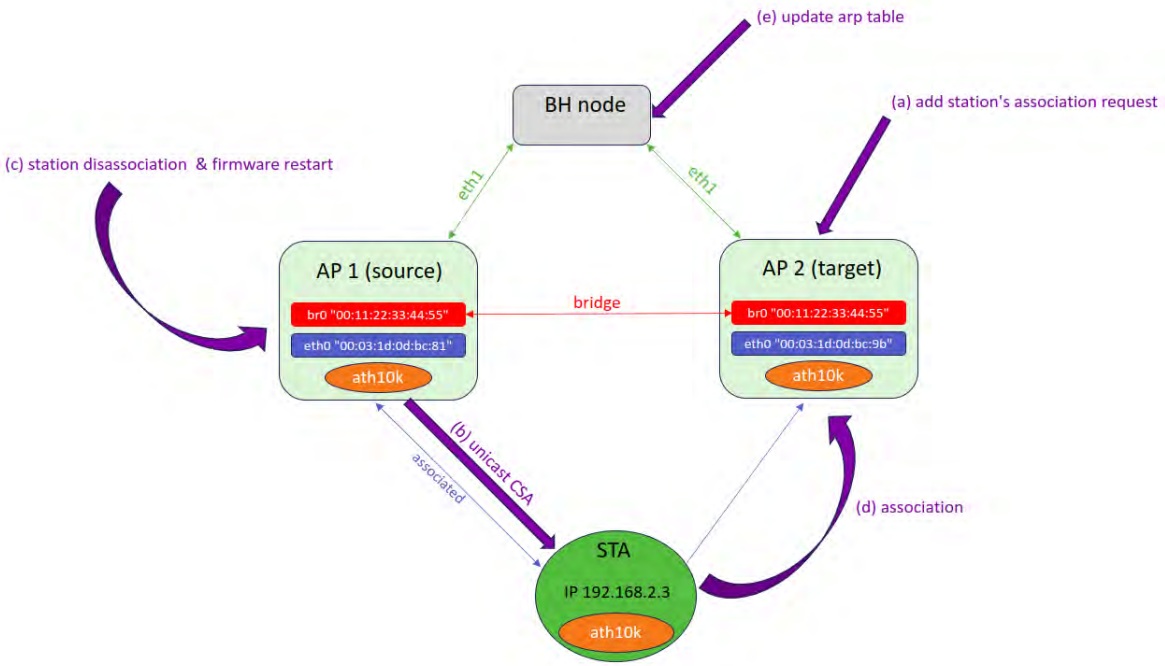

Figure 3.1: Client steering mechanism

Chapter 4

Reinforcement Learning

4.1 Introduction

Machine learning is an enlarged field that contains a variety of techniques for building intelligent systems. Although Reinforcement Learning is a sub field of machine learning which involves, as many other techniques, training algorithms to learn from data there are some key differences between them.

In other types of machine learning, like supervised learning, algorithms are trained on labeled data sets that specify the correct output for each input. Instead Reinforcement Learning involves learning in an interactive environment with trial and error approach, where the feedback may be delayed and noisy. In addition other types of machine learning, such as unsupervised learning, may not include explicit decision making. Reinforcement Learning however usually involves sequential decision making where the actions of an agent affect the state of the environment, which afterwards affects the future reward that agent will receive. Generally, Reinforcement Learning as technique is chosen in cases where the optimal solution is not sufficiently defined or changes with time, so the agent must constantly explore the environment and adjust its behaviour in order to maximize the expected reward.

In summary, Reinforcement Learning is a powerful approach in designing intelligent systems which are deployed in dynamic and uncertain environments and there are capable to learn from their own experience.

4.2 Deep Q Learning

Deep Q-Learning is a reinforcement learning technique for estimating the optimal action-value function in a Markov Decision Process (MDP), which represents the expected reward for taking a specific action in a specific state so following the optimal policy afterwards.

In Deep Q-Learning a neural network, known as a Q-network, takes the state of the environment as input and outputs the expected reward for each possible action. During training Q-network is updated by using a variation of bellman equation, which relates recursively the value of a state-action pair to the expected value of the next state-action pair. However, one drawback of Deep Q-Learning is that it can be unstable and prone to overestimating the value of actions. Several modifications, such as Double Q-Learning and Prioritized Experience Replay, have been proposed to address these issues and improve the stability and convergence of Deep Q-Learning.

In conclusion, Deep Q learning algorithms are capable of playing complex games and controlling robotic systems by combing the the power of deep neural networks with the reinforcement learning framework. The list of successful application of DQN is expanding in to a wide range, including playing Atari games, controlling robots, and optimizing energy consumption in data centers.

4.3 Deep Q Learning approach in WiFi association

In WiFi topologies with multiple access points using Deep Q Learning for station association can improve performance and accuracy as well as dynamic adaptation and scalability. However, there are also potential drawbacks that must be carefully considered before implementing this technique. So in the following sections we will discus the benefits and the potential drawbacks of applying RL and DQN in order to optimize the association process in WiFi topologies with multiple access points.

4.3.1 802.11 Association Frame

The default association process for a WiFi station in a topology with multiple access points is relatively simple and is typically based on the signal strength of the access points. This process is commonly referred to as the ”best signal” algorithm, as it simply selects the

access point with the strongest signal. Particularly, when a WiFi station searches for an access point to associate with it will scan the area and collect information on the available access points, including their signal strength, frequency, and other relevant details. Then the station will evaluate the available access points and choose the one with the strongest signal strength. While this process can be effective in many situations, it may not always result in the optimal choice of access point, especially in complex environments with multiple access points and competing factors, where may prove suitable more advanced algorithms, such as Deep Q Learning, as they can take into account a broader range of factors and make more informed association decisions.

4.3.2 802.11ac Association Criterias

In the 802.11ac protocol, the process of access point selection when multiple access points are available is based on the concept of Basic Service Sets (BSS), which is the fundamental building block of the 802.11 wireless network, and in the use of the 802.11k and 802.11v standards.

Specifically, the basic service set (BSS) is a set of stations (e.g., clients and access points) that communicate with each other directly and in a multiple access point environment each access point forms its own BSS. In order to select the best access point, a WiFi station scans the available BSSes and collects information on various parameters such as signal strength, channel quality, and data rate support, which is provided by the access points using the 802.11k standard. Consequently evaluates the collected information and selects the best access point based on the following criteria:

1. Received Signal Strength Indicator (RSSI): This metric measures the signal strength between the station and the access point.

2. Channel Quality Indicator (CQI): This metric measures the quality of the channel between the station and the access point.

3. Data Rate Support: This metric measures the maximum data rate supported by each access point.

4. Load Balancing: This metric balances the load across the available access points to prevent any one access point from becoming overloaded. The station selects the access point with the lowest number of connected clients.

5. Quality of Service (QoS): This metric ensures that the selected access point can provide the required QoS for the station's traffic.

Also the access point is able to provide additional information to the station, such as it's capabilities and available services, with the 802.11v standard. The process of access point selection in the 802.11ac protocol is a more sophisticated and comprehensive approach than the "best signal" algorithm used in older protocols because it takes into account multiple factors to make a more informed decision and provide a better user experience.

4.3.3 DQN comparison with default association frame

Using Deep Q Learning to predict the best association access point in a WiFi topology with multiple access points offers several benefits compared to the 802.11ac protocol criteria.

1. Flexibility: Deep Q Learning can take into account a broader range of factors than the 802.11ac protocol criteria. For example, it can consider the number of clients connected to each access point, the available bandwidth, and the traffic congestion on each access point. This flexibility allows Deep Q Learning to adapt to changing network conditions and make more informed decisions.

2. Optimization: Deep Q Learning can optimize the association decision for a specific user or group of users, rather than just selecting the access point with the best signal strength or channel quality. This can lead to a better user experience, particularly in complex environments with multiple access points and competing factors.

3. Learning: Deep Q Learning can learn from past experiences and improve its association decisions over time. This means that it can adapt to changes in network conditions and make more informed decisions based on past successes and failures.

4. Adaptability: Deep Q Learning can adapt to different network environments and scenarios, whereas the 802.11ac protocol criteria are based on fixed parameters that may not be suitable for all situations.

5. Performance: Deep Q Learning can improve the overall performance of the network by balancing the load across the available access points and reducing congestion. This can lead to better network efficiency and faster data transfer rates.

While the 802.11ac protocol criteria are effective in many situations, they have limitations and may not always result in the optimal choice of access point. Deep Q Learning can overcome some of these limitations and provide a more advanced and comprehensive approach to access point selection.

However, there are also some potential drawbacks to using Deep Q-Learning for WiFi association:

1. Training Data: Deep Q-Learning requires a large amount of training data to learn effectively, which can be challenging to collect in WiFi environments.

2. Complexity: Deep Q-Learning is a complex algorithm that requires specialized expertise to implement and optimize effectively.

3. Resource Requirements: Deep Q-Learning can require significant computational resources, especially when dealing with large and complex WiFi topologies.

In summary, using Deep Q-Learning for WiFi association in topologies with multiple access points can lead to improved performance and accuracy, as well as dynamic adaptation and scalability. However, there are also potential drawbacks that must be carefully considered before implementing this technique.

4.4 DQN algorithm presentation

The DQN algorithm is written in Python 3.8.10 and for the implementation of the neural network we use the Tensorflow library (tensorflow 2.11.0 version). In order to create the data set which is used for training the algorithm we use the Pandas library. In our approach to model the association problem that shows up in an WiFi network with multiple stations and multiple access points we introduce the following classes: STA, AP, $ApEnv$ and $DQNAgent$.

4.4.1 class STA

This class is used to monitor each station's state. So we store the mac of the station, the mac of the access point associated to, the requested throughput of the station and the achieved throughput when is connected to the network.

```
class STA:
    def __init__(self, mac, AP_mac):
        self.mac = mac
        self.AP_mac = AP_mac
        self.req = 0
        self.ach = 0

    def set_req(self, req):
        self.req = req

    def set_ach(self, ach):
        self.req = ach

    def set_AP_mac(self, ap_mac):
        self.AP_mac = ap_mac

    def get_req(self):
        return self.req

    def get_ach(self):
        return self.ach

    def get_AP_mac(self):
        return self.AP_mac
```

Listing 4.1: class STA

4.4.2 class AP

In order to capture the state of the access point we store its mac address and a list of the stations connected to it. The list is updated dynamically with the functions add_sta and rm_stas.

At every step the machine learning algorithm makes a choice corresponding to the previous state of the network. Every choice is related to a certain action for each access point, which basically is the movement of a station from one access point to one other. The space of actions contains twelve actions because in our modeling the network consists of tow access points and the maximum number of stations that can be connected in each of them is three.

So for each station there are tow possible actions, e.g. one for moving to each access point. The same action value is passed as an argument in function $action$ for both access points at every step and each one decides how to act. More specific if the actions declares a station movement towards the access point the target station is added in the access point's station list and the function $move$ is called in order to make the actual movement of the station in the network. If the the station is already connected nothing happens and finally if the station has to move to the other access point it is removed from the station's list without calling the $move$ function because it is only called once from the target AP.

In the context of this thesis only to channels are available in the 5GHz band. One transmits at the channel 36 with central frequency 5180 MHz and the other at channel 149 with central frequency 5745 MHz. The client steering mechanism which has been described in the previous chapter is triggered from the move function. The DQN algorithm runs remotely in one of the NITOS testbed nodes and the node does not have access to the name space of the other nodes where the access points are hosted. So the $client\_steering.sh$ script must be called from the server which have access to the name space of all nodes. In order to achieve synchronization between the execution in the server and the python script of the algorithm we use $sshfs$ command to mount remote file system over ssh. In that way we use a common file to create a semaphore alike functionality. It is important to notice that we can run the DQN algorithm remotely, for example in our local computer, and is not necessary to be placed in of the nodes. More detailed explanation of the experimental topology and the NITOS testbed will follow in the next section.

The function new_sta is used to check if new stations have been connected to the AP during the time interval between at every algorithm's step. If a new station is detected it is added in the station's list but it will be declared as not associated until the first association function that refers to it is taken. More about the process of detection a new station will be explained also in the following section.

```
class AP:
    choice_dict = {0: 1, 1: 2, 2: 1, 3: 2, 4: 1, 5: 2,
                   6: 1, 7: 2, 8: 1, 9: 2, 10: 1, 11: 2}

    def __init__(self, mac, num):
        self.mac = mac
        self.num = num
```

```
        self.sta_list = []

    def add_sta(self, sta):
        sta.AP_mac = self.mac
        self.sta_list.append(sta)

    def rm_sta(self, sta):
        self.sta_list.remove(sta)

    def is_connected(self, sta_mac):
        path = "../state_request/panda_stats/state_request_AP" + str(self.num)
        df = pd.read_pickle(path)
        for column in df:
            if sta_mac in column:
                return True
        return False

    def action(self, choice, all_stas):
        if choice < (len(all_stas) * MAX_AP):
            target_num = self.choice_dict [choice]
            if target_num == self.num :
                if all_stas[int(choice/MAX_AP)] not in self.sta_list:
                    self.add_sta(all_stas[int(choice/MAX_AP)])
                    if not self.is_connected(all_stas[int(choice/MAX_AP)].mac):
                        self.move(all_stas[int(choice/MAX_AP)].mac, self.mac)
            else:
                if all_stas[int(choice/MAX_AP)] in self.sta_list:
                    self.rm_sta(all_stas[int(choice/MAX_AP)])

    def move(self, sta_mac, ap_mac):
        chan1 = 0
        chan2 = 0
        if self.num == 1:
            chan1 = 149
            chan2 = 36
        else:
```

```
            chan1 = 36
            chan2 = 149

        steering_command = "bash client_steering.sh " + str(chan1) + " " + str(chan2) + " " + str(sta_mac)
        with open('semaphore/sem1', 'w') as f:
            f.write(steering_command)

        while(os.stat("semaphore/sem1").st_size != 0):
            time.sleep(1)

        return None

    def new_sta(self, all_stas):
        sta_pend_list = []
        path = "../state_request/panda_stats/new_sta_AP" + str(self.num) + ".txt"
        with open(path, 'r') as file:
            lines = file.readlines()
        for line in lines:
            if "-" in line:
                spl = line[8:].split("-")
                if((len(spl) > 1) and (len(all_stas) < (MAX_AP * MAX_STAS_PER_AP))):
                    for i in range(0,len(spl)-1):
                        sta_pend_list.append(STA(spl[i], None))
                    return(sta_pend_list)
        return None
```

Listing 4.2: class AP

4.4.3 class ApEnv

In our approach for modeling the environment of the wireless network we create a list of STA objects (see all_stas at line 15) and we also define a dictionary with hard coded the mac addresses of the network's access points. The only assumption we make is that we know the ip address of access point's wireless interfaces and so the mac addresses could be found by requesting the python server hosted in each access point. More details for the python

client-server topology will be given below and in the following chapters.

The most important and obviously the larger function of this class is the $step$ function. As an argument in the function we pass the action that has been selected from the DQN algorithm. The structure of the function can be summarized in the following parts:

1. Current state: At first we request every access point's server for new connected stations and if any new stations are detected we append them in the all_stas list. Then we request again every access point in order to form the current state of the network. Each AP responds with a pandas Dataframe containing every associated station's mac address, requested throughput and achieved throughput. The request is made by executing the $app - client.py$ with $subprocess$ python library. Ass arguments we pass the ip of the server, the port that has been predefined and the name of the request. Based on the Dataframes collected we update the all_stas list with $update\_stas\_list$ function. Finally we make a copy of the current state which will be used later in order to determine the reward for the given action in that step.

2. Action: We call the $action$ function for each AP

3. Next state: We repeat the process described in current state. But instead of copying the station list we create an array of integers with size same as the stations of the network to model the stations associations. So for every station we put -1 if not associated yet by a valid action, 1 if associated in the first AP and 2 if associate din the second AP.

4. Calculate reward: For every action taken we calculate reward with the following method.

$$reward = 0.35 * weighted\_sum + 0.40 * avg\_throughput\_ratio + 0.25 * total\_ach\_thr\_factor$$

   By adjusting the weights in front of $weighted\_sum$ and $avg\_throughput\_ratio$, we can control the balance between prioritizing individual station throughput and load balancing. Also, the $total\_ach\_thr\_factor$ is used to prioritize over all greater channel utilization. The way that the above metrics are calculated is shown in the code below at lines 96-100. At last but not least we add a negative reward in two cases. The first is when an action is selected that does not corresponds to any station in cases where less than six stations (i.e the maximum number of stations) are connected. The second is

when not associated stations exist waiting to connect and the action which is selected corresponds to an already connected station, so the number of pending stations remains the same. This negative reward aims to force the agent to immediately associate new stations.

5. Check for Terminal state: As terminal state we consider the completion of 15 episodes if no satisfying enough reward is accomplished earlier. The reward satisfaction criteria is basically the maximization of reward. The maximum value of reward is 1 but is not possible to be achieved at every topology. For that reason we define a theoretical threshold (see $reward\_norm$ at line 115) for the ideal reward considering the theoretical capacity of the network and the total requested throughput for each topology. So we accept as terminal state a state where is evaluated with reward greater than the $reward\_norm$ or where 95% of the capacity of the network is utilized. Also all stations must have been associated in order a state to be considered as terminal.

```
class ApEnv:
    MOVE_PENALTY = 1
    PENDING_STA_PENALTY = 50
    ACTION_SPACE_SIZE = MAX_STAS_PER_AP*MAX_AP*MAX_AP

    AP1 = 1  # AP1 key in dict
    AP2 = 2  # AP2 key in dict

    d = {1: "00:03:1d:0d:bc:81",
         2: "00:03:1d:0d:bc:9b"}

    choise_dict = {0: 1, 1: 2, 2: 1, 3: 2, 4: 1, 5: 2,
                   6: 1, 7: 2, 8: 1, 9: 2, 10: 1, 11: 2}

    all_stas = []

    def __init__(self):
        self.all_stas = self.all_stas
        self.AP1 = AP(self.d[self.AP1], self.AP1)
        self.AP2 = AP(self.d[self.AP2], self.AP2)

    def reset(self):
```

```
        self.episode_step = 0
        self.all_stas = []
        observation = np.zeros(MAX_STAS_PER_AP * MAX_AP)

        return observation

def step(self, action):
        pending_stas = ""
        subprocess.run(["python3", "../state_request/app-client.py", "192.168.2.2", "5800", "search", "new_sta"])
        subprocess.run(["python3", "../state_request/app-client.py", "192.168.2.4", "5900", "search", "new_sta"])
        new_sta = self.AP1.new_sta(self.all_stas)
        accepted = "04:f0:21:28:a9:74" + "04:f0:21:28:a9:6f" + "04:f0:21:28:a9:a1" + "04:f0:21:2c:6d:d1" + "04:f0:21:28:a9:94" + "04:f0:21:25:1f:f0"
        if new_sta is not None:
            for st in new_sta:
                if (st.mac in accepted) and (str(st.mac) not in pending_stas):
                    pending_stas += (str(st.mac) + " ")
                    self.all_stas.append(st)
        new_sta = self.AP2.new_sta(self.all_stas)
        if new_sta is not None:
            for st in new_sta:
                if (st.mac in accepted) and (str(st.mac) not in pending_stas):
                    pending_stas += (str(st.mac) + " ")
                    self.all_stas.append(st)

        subprocess.run(["python3", "../state_request/app-client.py", "192.168.2.2", "5800", "state_request"])
        subprocess.run(["python3", "../state_request/app-client.py", "192.168.2.4", "5900", "state_request"])
        df1 = pd.read_pickle("../state_request/panda_stats/state_request_AP1")
        df2 = pd.read_pickle("../state_request/panda_stats/state_request_AP2")
        df_total = pd.concat([df1,df2], axis=1)
```

```
        self.update_stas_list(df_total, self.all_stas)
        old_observation = copy.deepcopy(self.all_stas)
        self.episode_step += 1

        self.AP1.action(action, self.all_stas)
        self.AP2.action(action, self.all_stas)

        subprocess.run(["python3", "../state_request/app-client.py", "192.168.2.2", "5800", "state_request"])
        subprocess.run(["python3", "../state_request/app-client.py", "192.168.2.4", "5900", "state_request"])
        df1 = pd.read_pickle("../state_request/panda_stats/state_request_AP1")
        df2 = pd.read_pickle("../state_request/panda_stats/state_request_AP2")
        df_total = pd.concat([df1,df2], axis=1)

        self.update_stas_list(df_total, self.all_stas)

        new_observation = np.zeros(MAX_STAS_PER_AP * MAX_AP * 3)
        for i in range(0,len(self.all_stas)):
            if self.all_stas[i].AP_mac == None:
                new_observation[i*3] = -1
            else:
                new_observation[i*3] = self.reverse_lookup(self.d, self.all_stas[i].AP_mac)
                new_observation[i*3+1] = self.all_stas[i].req
                new_observation[i*3+2] = self.all_stas[i].ach

        not_associated_stas = 0
        reward = 0
        total_req_thr = 0
        total_ach_thr = 0
        for i in range(0, len(self.all_stas)):
            total_req_thr += self.all_stas[i].req
            total_ach_thr += self.all_stas[i].ach
            if self.all_stas[i].AP_mac == None:
                not_associated_stas += 1
```

```
86
87          if action >= (len(self.all_stas) * MAX_AP):
88              if len(self.all_stas) == 0:
89                  reward = 0
90              else:
91                  reward = -100
92          else:
93              if old_observation[int(action/MAX_AP)].AP_mac != None and
    not_associated_stas > 0:
94                  reward = -50
95              else:
96                  num_stations = len(self.all_stas)
97                  # Calculate achieved throughput ratio for each station
98                  throughput_ratios = [self.all_stas[i].ach / self.all_stas
    [i].req for i in range(num_stations)]
99                  # Calculate weight for each station based on requested
    throughput
100                 weights = [self.all_stas[i].req / total_req_thr for i in
    range(num_stations)]
101                 # Calculate average achieved throughput ratio
102                 avg_throughput_ratio = sum(throughput_ratios) /
    num_stations
103                 # Calculate weighted sum of achieved throughput ratios
104                 weighted_sum = sum([throughput_ratios[i] * weights[i] for
     i in range(num_stations)])
105                 # Calculate total achieved throughput factor
106                 total_ach_thr_factor = float(total_ach_thr/(MAX_AP *
    MAX_AP_THR))
107                 # Combine average throughput ratio and weighted sum with
    appropriate weights
108                 reward = float(100)*(0.25 * weighted_sum + 0.25 *
    avg_throughput_ratio + 0.5 * total_ach_thr_factor)
109
110         done = False
111         if total_req_thr == 0:
112             reward_norm = 0.85*100
113         else:
114             reward_norm = (1/float(total_req_thr/(MAX_AP * MAX_AP_THR)))
    *100
```

```
            if reward_norm < 0.85*100:
                reward_norm = 0.85*100

        if (self.episode_step >= 15) or ((len(self.all_stas) == MAX_AP*MAX_STAS_PER_AP) and ((total_ach_thr >= 0.95*MAX_AP*MAX_AP_THR) or (reward >= reward_norm)) and (not_associated_stas == 0)):
            done = True

        return new_observation, reward, done

    def update_stas_list(self, df, sta_list):
        for i in range(0,len(sta_list)):
            for column in df:
                if sta_list[i].mac in column:
                    if sta_list[i].AP_mac != None:
                        if "_req" in column:
                            sta_list[i].set_req(df[column].iloc[0])
                        if "_ach" in column:
                            sta_list[i].set_ach(df[column].iloc[0])
                        if "_AP" in column:
                            sta_list[i].set_AP_mac(df[column].iloc[0])

    def reverse_lookup(self, d, v):
        for k in d:
            if d[k] == v:
                return k
        return None
```

Listing 4.3: class ApEnv

4.4.4 class DQNAgent

The neural network is created using the tf.keras API, which is a high-level interface for building and training deep learning models using TensorFlow. The neural network architecture consists of four fully connected layers:

1. Input layer which expects a 1D array of length $MAX\_STAS\_PER\_AP*MAX\_AP*3$. For every connected station we take as input three parameters: AP associated with,

requested throughput and achieved throughput. So for six station (i.e maximum number of stations) in our implementation we have eighteen units as input layer.

2. Dense layer with 16 units and relu activation function. This layer is the first hidden layer in the neural network.

3. Dense layer with 16 units and relu activation function. This layer is the second hidden layer in the neural network.

4. Dense layer with $MAX\_STAS\_PER\_AP*MAX\_AP*MAX\_AP$ units and linear activation function. This layer is the output layer of the neural network and uses the linear activation function. The output of this layer is the predicted Q-values for each possible action. So for six stations we have twelve action and the same number of output layer units.

The relu activation function (short for ”rectified linear unit”) is a common activation function used in deep neural networks. It is defined as relu(x) = max(0, x), which means that it returns the input value if it is positive, and 0 otherwise. The linear activation function simply returns the input value unchanged. By using a neural network with multiple layers and nonlinear activation functions, the DQNAgent is able to learn a more complex mapping from states to Q-values, which can improve its ability to make accurate predictions and learn to play the game more effectively.

The train function is responsible for training the Deep Q-Network by updating its weights based on the loss and accuracy values obtained during the training process. The function takes in two parameters: $terminal\_state$ and $step$. The $terminal\_state$ parameter is a boolean value that indicates whether the current state is a terminal state or not. A terminal state is a state where the episode has ended, and there are no further actions to be taken. The step parameter, on the other hand, keeps track of the number of steps taken so far during the training process. The function first checks if the number of samples in the replay memory is greater than the minimum replay memory size specified by $MIN\_REPLAY\_MEMORY\_SIZE$. If it is not, the function returns without performing any training. If the minimum replay memory size is met, the function randomly samples a minibatch of transitions from the replay memory with a size of $MINIBATCH\_SIZE$. The current and future states and the corresponding action, reward, and done flag for each transition are extracted from the minibatch. Next, the function preprocesses the current and future states to ensure they are in the appropriate format

for the neural network. It then predicts the Q-values for the current and future states using the main and target networks, respectively. The function then constructs the training data, which consists of the current states and the corresponding updated Q-values. The updated Q-values are calculated based on the Bellman equation and the target Q-values predicted by the target network. Eventually, we converge the two models so they are the same, but we want the model that we query for future Q values to be more stable than the model that we're actively fitting every single step.

Finally, the function trains the main network using the training data and logs the loss and accuracy values to TensorBoard if the current state is a terminal state. If the target update counter reaches the specified $UPDATE\_TARGET\_EVERY$ value, the function updates the target network with the weights of the main network.

```
class DQNAgent:
    def __init__(self):
        self.model = self.create_model()
        self.target_model = self.create_model()
        self.target_model.set_weights(self.model.get_weights())

        if LOAD_MODEL is not None:
            self.replay_memory = pickle.load(open('models/buffer2.pkl', 'rb'))
        else:
            self.replay_memory = deque(maxlen=REPLAY_MEMORY_SIZE)

        self.tensorboard = ModifiedTensorBoard(log_dir="logs/{}-{}".format(MODEL_NAME, int(time.time())))
        self.target_update_counter = 0

    def create_model(self):
        if LOAD_MODEL is not None:
            model = tf.keras.models.load_model(LOAD_MODEL)
        else:
            model = tf.keras.models.Sequential()
            model.add(tf.keras.layers.Dense(units=16, activation='relu', input_shape=(MAX_STAS_PER_AP*MAX_AP*3,)))
            model.add(tf.keras.layers.Dense(units=16, activation='relu'))
            model.add(tf.keras.layers.Dense(units=(MAX_STAS_PER_AP*MAX_AP*MAX_AP), activation='linear'))
```

```
23
24          model.compile(loss="mse", optimizer=Adam(lr=0.001), metrics=[
    'accuracy'])
25      return model
26
27  def update_replay_memory(self, transition):
28      self.replay_memory.append(transition)
29
30  def train(self, terminal_state, step):
31      if len(self.replay_memory) < MIN_REPLAY_MEMORY_SIZE:
32          return
33
34      minibatch = random.sample(self.replay_memory, MINIBATCH_SIZE)
35
36      current_states = np.array([transition[0] for transition in
    minibatch])/MAX_AP
37      current_qs_list = self.model.predict(current_states)
38
39      new_current_states = np.array([transition[3] for transition in
    minibatch])/MAX_AP
40      future_qs_list = self.target_model.predict(new_current_states)
41
42      X = []
43      y = []
44
45      for index, (current_state, action, reward, new_current_state,
    done) in enumerate(minibatch):
46          if not done:
47              max_future_q = np.max(future_qs_list[index])
48              new_q = reward + DISCOUNT * max_future_q
49          else:
50              new_q = reward
51
52          current_qs = current_qs_list[index]
53          current_qs[action] = new_q
54
55          # And append to our training data
56          X.append(current_state)
57          y.append(current_qs)
```

```
        self.model.fit(np.array(X)/MAX_AP, np.array(y), batch_size=MINIBATCH_SIZE, verbose=0, shuffle=False, callbacks=[self.tensorboard] if terminal_state else None)

        if terminal_state:
            self.target_update_counter += 1

        if self.target_update_counter > UPDATE_TARGET_EVERY:
            self.target_model.set_weights(self.model.get_weights())
            self.target_update_counter = 0

    def get_qs(self, state):
        return self.model.predict(np.array(state).reshape(-1, *state.shape)/MAX_AP)[0]
```

Listing 4.4: class DQNAgent

Replay Memory

In Reinforcement Learning the technique of Replay memory is used in order the training of a neural network to become more effective and stable. According to Reinforcement Learning an agent learns to take actions in an environment to maximize a reward signal and interacts with the environment by state observation, taking an action, receiving a reward and transition to a new state. During training through interaction, the agent updates his strategy based on the experiences he gained through interaction, which are stored in a buffer called Replay memory. The agent takes samples of a small batch of experiences from Replay memory in order to update its policy. Below are listed some of the major advantages of Replay memory:

1. Improved sample efficiency: As the agent can learn from past experiences the need for additional interactions with the environment is limited so saving time and resources is achieved.

2. Increased stability: By sampling experiences randomly from the replay memory, the agent is less likely to over fit to recent experiences and can explore better the environment.

3. Better use of data: The agent uses the stored information in order to update its policy multiple times, increasing the number of updates per experience

Minibatch trainning

Minibatch training is a common machine learning technique for improving the training of deep learning models, including neural networks. Specifically, Minibatch refers to a small subset of the training data used for training of a model in a single iteration of the optimization algorithm. During minibatch training, the training data is divided into small batches and the model parameters are updated based on the average of the gradients computed on the samples in the minibatch. The size of minibatch is a hyper parameter that must be set based on available memory and the complexity of the model.

Using mini-batches has many advantages, including:

1. Reduced memory requirements: This kind of training demands less memory compared to batch training which makes it more appealing for large data sets that cannot be stored in memory.

2. Faster convergence: The convergence of the optimization algorithm is faster than using a large batch due to the more frequent updating model parameters with smaller batches.

3. Better generalization: The randomness introduced in the learning process with mini-batch training approach can help the model generalize

Chapter 5

Experimental Tools

5.1 Introduction

In this section we will describe the most important instruments which we use to evaluate the experimental implementation. These are the following:

1. NITOS, a wireless testbed located in University of Thessaly, Greece.
2. ath10k driver, an open-source Linux kernel driver for Atheros 802.11ac wireless LAN chips.
3. Iperf3, a widely used command-line tool for measuring network performance.
4. Wireshark, an open-source network protocol analyzer used to capture and analyze network traffic.

5.2 NITOS testbed

The Center for Research Technology Hellas (CERTH) is associated with the Network Implementation Testbed Laboratory of the Department of Electrical and Computer Engineering at University of Thessaly (NITlab). The design, analysis, and implementation of wireless and wired methods, as well as how well they perform in actual environments, are the main topics of the lab's research. NITlab has created a facility called NITOS, short for Network Implementation Testbed utilizing Open Source platforms, in this context[10].

The NITOS facility, which presently has more than 100 active wireless nodes, was built to facilitate protocol and application evaluation in real-world contexts while simultaneously

achieving reproducibility of experimentation. Three deployments of the NITOS facility are separated by geography. The University of Thessaly (UTH) campus building has three testbeds: an outdoor one on the building's exterior, an indoor one in the basement, and an office testbed set up in CERTH's office building in Volos[10]. The cOntrol and Management Framework (OMF) open-source software is used to control and manage the facility. By reserving slices (nodes, access points, base stations, or frequency spectrum) of the testbed through the NITOS scheduler, which works in conjunction with OMF to promote ease of use for experimentation, users can conduct their experiments[10].

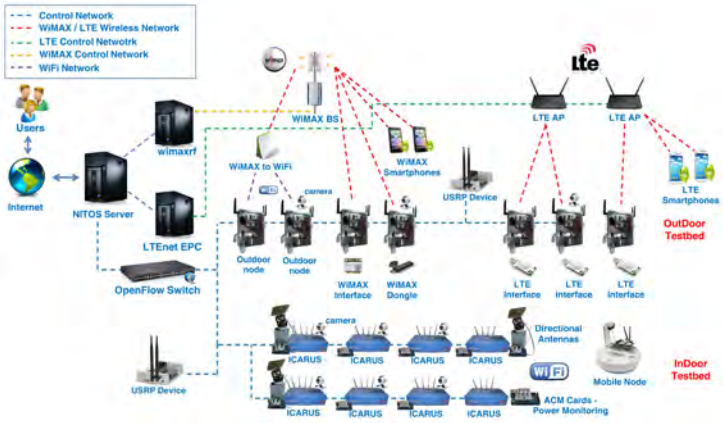

Figure 5.1: Nitos Architecture

5.2.1 Outdoor Testbed

Powerful nodes used in the NITOS Outdoor deployment have numerous wireless interfaces and enable testing of heterogeneous (Wi-Fi, WiMAX, LTE) wireless technologies. It has 50 strong nodes and is installed on the outside of a University of Thessaly (UTH) campus building, as seen in the following figure. The nodes are spread out throughout the building's roof, 25 of them in a grid topology, and 25 of them on its floors[11].

5.2.2 Indoor Testbed

The NITOS RF Isolated Indoor Deployment is set up inside one of the University of Thessaly's campus buildings and consists of 50 Icarus nodes with Wi-Fi, WiMAX, and LTE wireless interfaces. In a big testbed, experimenters can run and assess power-demanding processing algorithms and protocols. Additionally, it has directional antennas and other cutting-edge prototypes. The NITOS indoor testbed's isolated environment is surrounded by sym-

Figure 5.2: Nitos Outdoor testbed

metrically positioned Icarus nodes, creating a grid topology. The nodes' distance from one another is fixed at 1.2 meters, and they are all the same height. As a result, a homogeneous environment with isobaric nodes and equal capabilities is produced[12].

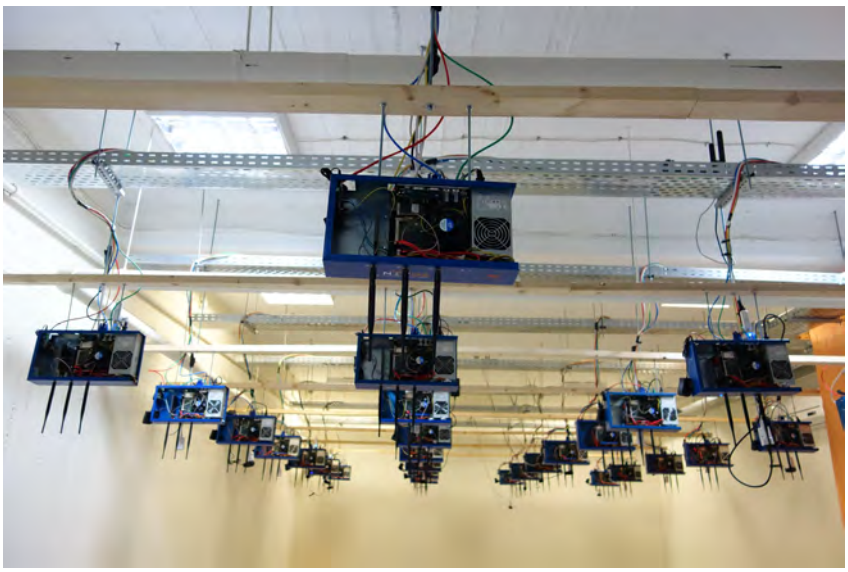

Figure 5.3: Nitos Indoor testbed

5.2.3 Office Testbed

Ten strong Icarus nodes of the second generation make up the Office Indoor Testbed. The nodes contain a variety of heterogeneous technologies, including WiMAX, LTE, and WiFi, and they enable experimenters to create and carry out realistic scenarios in a deterministic office setting[10].

Figure 5.4: Nitos Office testbed

5.3 ath10k driver

Ath10k is an open-source Linux kernel driver for Atheros 802.11ac wireless LAN chips, which was originally developed by Qualcomm Atheros and was later open-sourced to the Linux community. It provides support for the QCA988x and QCA6174 series of wireless chips and supports both access point (AP) mode and station (STA) mode. Ath10k uses the mac80211 subsystem of the Linux kernel to communicate with the wireless hardware and is compatible with various advanced features, including 802.11ac MU-MIMO, beamforming, and packet injection. Also supports regulatory domain and channel settings in order to comply with regional regulations regarding wireless networking. Instead of relying solely on the CPU ath10k supports firmware offloading, which allows the wireless hardware to handle some of the processing tasks, such as encryption and decryption, improves performance and reduces CPU usage leading to better overall system performance. Ath10k driver is maintained as part of the Linux kernel, and it is actively developed and updated by the Linux community while those characteristics enhanced to be widely used in various wireless networking devices, including routers, access points, and wireless network cards.

5.4 Iperf3

Iperf3 is a command-line tool for measuring network performance and is the successor to the original iperf and iperf2 tools. Expanding the capabilities of the previous versions iperf3 is designed to provide a more modern and feature-rich approach to measuring network bandwidth, throughput, and various network parameters. One very important application of iperf3, which is also adopted in our implementation, is to perform network performance tests

between two endpoints, typically a client and a server. It supports both TCP and UDP protocols and offers a range of options to customize the test parameters. Some other common use cases for iperf3 include network troubleshooting, network capacity planning, and performance benchmarking.

5.5 Wireshark

Wireshark is an open-source network protocol analyzer that is able to capture and analyze network traffic in real-time or by uploading saved packet capture files, e.g captured files with use of $tcpdump$ command. Below we present some of the most important functionalities of Wireshark.

It supports a large number of protocols, including popular ones like TCP, UDP, HTTP, DNS, DHCP, SSL/TLS, and it can dissect and display the details of each packet according to the specific protocol. Wireshark also provides powerful filtering capabilities in order to examine specific packets or protocols of interest by creating complex display filters to search for specific criteria. One other feature is that provides detailed packet-level analysis, allowing user to inspect packet headers, payloads, track packet flows, and analyze packet timing. The fact that can dissect and decode different protocols, providing a human-readable representation of network traffic helps user to understand the structure and content of each packet, aiding in troubleshooting and analysis. It also supports a modular architecture with the installation of additional dissectors, plugins, and scripts. These extensions enhance its capabilities and enable analysis of specialized protocols or provide additional features. Finally, a feature which we exploit in our implementation in order to evaluate the network performance is statistical analysis capabilities of Wireshark, as it is able to generate various statistics and summaries based on captured packets such as protocol distribution, packet lengths, round-trip times, and more.

5.6 Chapter Conclusion

In this chapter we present the experimental tools as well as the infrastructure used to deploy the experimental WiFi network topology. Specifically, the WiFi topology that we will describe in the next chapter was deployed in Nitos Indoor RF Isolated Testbed where the

Icarus Nodes hosted the AP and STA entities of our topology. Both STAs and APs operate in 5GHz band and 802.11ac protocol using the ath10k driver. With iperf3 we send UDP traffic between a back-hole node and the STAs and with Wireshark we analyze the traffic that flows through the APs.

Chapter 6

Experimental WiFi Network Topology and DQN evaluation

6.1 Experimental WiFi Network Topology

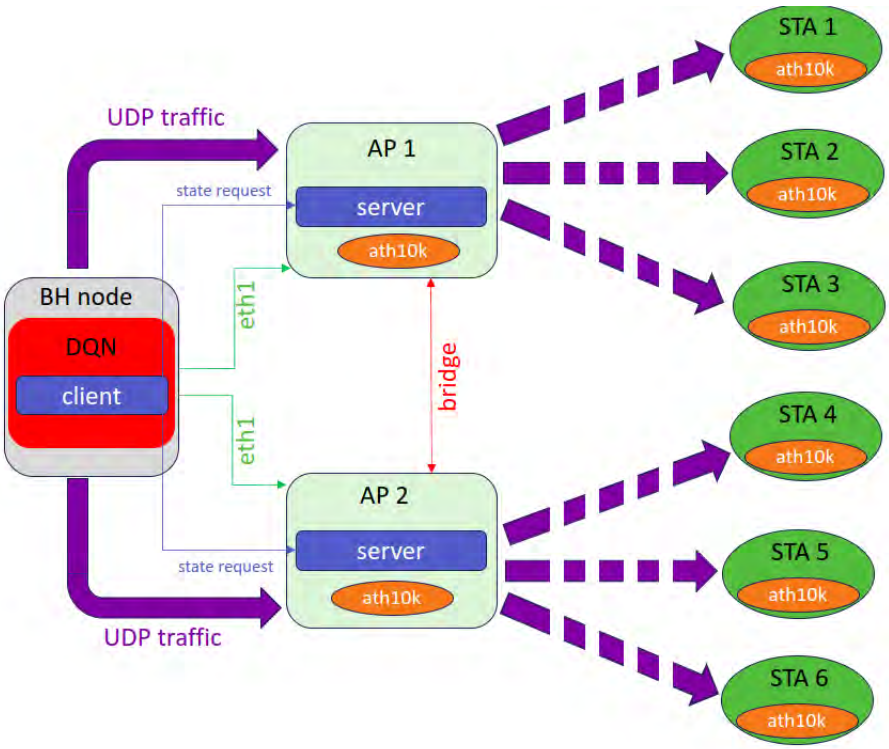

Figure 6.1: Experimental WiFi Network Topology

At chapter 3 a simplistic topology of tow access points and one station is described in order to explain the basic steps of the client steering mechanism. Now we will describe the

complete experimental topology that has been exploited in NITOS Indoor RF isolated testbed. The topology consists of two access points, six stations and one back haul node from whom we send ip traffic towards the stations. All of them, access points, stations and back haul node, are hosted on different testbed nodes. In both access points we set up bridge interfaces, which connects two different interfaces (bridge ports). Bridging two interfaces causes every Ethernet frame that is received on one bridge port to be transmitted to the other port. Thus, the two bridge ports participate in the same Broadcast domain. In our case the bridge is a connection of Ethernet interface of the AP with the Wireless interface. The wireless interface has the same mac for both access points but the Ethernet have different in order the back haul can direct the traffic properly among the access points.

All nodes are connected to the $nitlab3$ server with Ethernet interface (eth0) which gives as the ability to execute ssh and $sshfs$ commands as we describe in the client steering mechanism. In addition the testbed infrastructure provides each node with one other Ethernet interface (eth1). So by utilizing eth1 interface we send UDP traffic from the back haul node towards the access points targeting the stations. When then packets reach the access points are transmitted wireless to the stations. The two access points have the same $ssid$ and the same mac which is achieved with bridge mechanism and is the key of seamless handover between access points. In all access points and stations we deploy $ath10k$ driver in AP and STA mode respectively and we add a monitor interface over wlan0. Also we update arp table in back hole node when a station is re-associated so UDP traffic towards the station is forwarded to the access point that the station is now connected.

As we can see in the picture above in BH node we exploit the DQN algorithm and in each AP a python3 server. The DQN agent as we describe in chapter 4 sends request in every access point's server in order to be informed about networks state. There are to type of request, $new_sta_request$ and $state_request$. The server is capable of supporting binary or json content messages. The first request is a json content request because the server responds with the mac address of recently connected stations. Specifically each AP uses $tcpdump$ over monitor interface to capture probe request frames of stations. In order to connect to a specific AP each STA sends a probe request frame and receives a probe response frame. So each probe request captured corresponds to a newly connected station. The second $state_request$ is a binary request because the server responds with a Pandas Dataframe that contains access point's current state. The state of the AP is defined by the stations connected to it so in each

Dataframe the AP stores the mac address, the achieved and the requested throughput for every station connected to it. The process of collecting the above metrics for every station is described in the section below.

6.1.1 Data Collection

Data collection happens in driver level. More specifically, in the content of this thesis ath10k driver is used. The data that we take under consideration are either metrics related to the AP or metrics related to stations connected to it. The data come from the commands ”iw wlan0 survey dump”, ”iw wlan0 station dump” and the file ”/sys/kernel/debug/ieee80211/phy0/ath10k/fw_stats”. Data collection process is fully automated with the use of schell script and python3 scripts. The data are updated dynamically every one second. Below we cite a link for a small demo of the dynamical data collection process for a topology of one AP and two stations connected to it.
link: `https://www.youtube.com/watch?v=1ahY5kRNuWQ`

With the automated process of data collection we are able to include in the data set any of the metrics the iw commands and the fw_stats provide us. However not all of those metrics are capable to contribute in the prediction of the best association scenario. A key concept to introduce at this point is that of feature importance, which refers to techniques that assign ”weights” to input metrics in terms of their usefulness in predicting the target variable. For that reason, after we make an initial separation of the metrics, we keep only those that we consider important. As part of the work, this process was initially done manually since many of the metrics are not updated as expected by the driver. Since we ended up at a more limited number of metrics we performed an analysis to remove ”irrelevant” columns from the data set and with the same analysis we try to reduce the problem of multicollinearity. Therefore, we will settle on the following final list of metrics which we use for the ML algorithm as we will explain below.

1. Transmitted bytes (txb): Refers to STA’s transmitted bytes in the observation period of time. A list of values is formed one for every STA connected to the AP.

2. Transmitted packets (txp): Refers to STA’s transmitted packets in the observation period of time. A list of values is formed one for every STA connected to the AP.

3. Transmission Failures (txf): Refers to the number of packets STA failed to transmit in

the observation period of time. A list of values is formed one for every STA connected to the AP.

4. Channel active time (actt): Refers to the time in milliseconds of the observation period that the channel, where the AP is tuned and transmits, is active. Referring to the channel as active means that a transmission occurs in the channels central frequency.

The data set that DQN algorithm takes as input is updated dynamically once the DQN agent is exploited in the experimental topology. With the use of iperf3 and schell script we simultaneously start transmissions from the stations that implement the iperf3 clients to the BH node that implements the iperf server. In order to exploit different association scenarios at every scenario we connect all stations to the network in a time interval of 60 seconds. During that period of time the moment that each STA is connected is selected randomly as well as the requested throughput for each one. In this way we can create many different scenarios simulating real conditions of a network in order to create a sufficiently large and sufficient data set that will feed the ML algorithm.

As we mention in the chapter 4 in ML implementation for every station we measure the requested and achieved throughput in every topology. In order the access point to be informed about the demand of each station connected to it, i.e it's requested throughput, we advertise the throughput demand of every station exploiting tos feature in iperf3. More specifically we define 3 categories representing different levels of demand and each one of them corresponds to a specific tos value. So for each STA the higher the demand, the higher the value of tos selected. Below we present the 3 levels of demand based on tos values where max_thr is referred to the theoretical maximum throughput that the AP can provide to a single STA and num_of_stas to the number of stations connected to the access point in the moment of measurement.

1. Tos 0x10: Supports rates from 0 to 30% of max_thr and the proportional demand is max_thr/num_of_stas in MBps

2. Tos 0x38: Supports rates from 30% to 60% of max_thr and the proportional demand is 2*(max_thr/num_of_stas) in MBps

3. Tos 0x58: Supports rates from 60% to 100% of max_thr and the proportional demand is max_thr in MBps

The access point is informed about the demand of each station by capturing UDP packets and decoding tos field with the use of $tcpdump$ over monitor interface. So when a $new\_sta\_request$ arrives the tos values are correlated with the connected stations and a response messages is send back to the DQN client.

For measuring the achieved throughput for every connected STA the AP utilizes the driver metrics mentioned above. When a $state\_request$ is send by the DQN Agent the access point starts measurements which are made every 1 second with a total duration of 15 seconds and then achieved throughput for every STA is calculated and stored in the servers response Dataframe. The achieved throughput is calculated with the following formula.

$$Achieved\,throughput \quad = \quad \frac{txb * \frac{txp-txf}{txp}}{actt * 1000}\ in\ MBps \tag{6.1}$$

It is important to mention that the **fully automated creation of the data set** enables us to train the ML algorithm not only in advance but also during its implementation in a real system.

6.2 DQN training

The process of training the DQN agent is probably the most important and difficult part of our implementation. In order the algorithm to behave in an efficient and stable way must be trained in thousands of association scenarios and the reward function, which is explained in the previous section, must be well defined. Before concluding in the code that is presented in chapter 4 we examined different hyper parameters for the ML algorithm and different reward functions through experimental testing. Because of the large number of iterations which are necessary to create a large enough data set initially we trained the algorithm off-line in a simulation topology. We have the ability to save any model that satisfies are behaviour criteria and then to load it in the DQN agent. When the agent is exploited in the real implementation continues its training with real data since the form of the data set (columns of the data set) remains the same.

6.2.1 Off-line training

In our approach to make a simulation environment we made some changes in the classes AP and $ApEnv$ compared to the codes presented in chapter 3. More specifically we modified

the function new_sta in class AP, see code below, and now we randomly create new stations to associate with the access points with random throughput demand. It is important to mention that in simulation topology we have the same number of stations and access points with the real testbed topology (2 AP and 6 STA).

```
def new_sta(self, all_stas):
    if random.uniform(0,1) > 0.5:
        if len(all_stas) < (MAX_AP * MAX_STAS_PER_AP):
            st = STA("0:0:0:" + str(len(all_stas)+1), None)
            st.set_req(10*random.randint(1,7))
            st.set_ach = 0
            return st
    return None
```

Listing 6.1: new_sta function

Because in the simulation the network topology does not exist the Agent does not need to send request in the access points in order to be informed for the networks state. So the $step$ function in $ApEnv$ is modified and instead of the $update_stas_list$ function the $update\_state\_EXP$ function is called, which has the same arguments.

```
    def update_state_EXP(self, sta_list):
        ap1_count = 0
        ap2_count = 0

        for st in sta_list:
            if st.get_AP_mac() == self.AP1.mac:
                ap1_count += 1
            if st.get_AP_mac() == self.AP2.mac:
                ap2_count += 1

        if ap1_count == 0:
            ap1_count = 1
        if ap2_count == 0:
            ap2_count = 1

        for i in range(0, len(sta_list)):
            if sta_list[i].AP_mac == self.AP1.mac:
                if sta_list[i].req < (MAX_AP_THR/ap1_count):
                    sta_list[i].ach = sta_list[i].req
```

```
                else:
                    sta_list[i].ach = MAX_AP_THR/ap1_count
            if sta_list[i].AP_mac == self.AP2.mac:
                if sta_list[i].req < (MAX_AP_THR/ap2_count):
                    sta_list[i].ach = sta_list[i].req
                else:
                    sta_list[i].ach = MAX_AP_THR/ap2_count
```

Listing 6.2: update_state_EXP function

As you can see in the code above, we set a theoretical available throughput capability for each access point and we make the assumption that the bandwidth is evenly distributed among the stations connected to it. In other words all stations of a specific access point achieve the same throughput. This approach is not realistic but is good enough to start training algorithm in basic but very important scenarios that are common. For example one of them is to associate new stations immediately to an access point not leaving them in pending state for a lot of iterations. We train the DQN agent over 3000 episodes and in each episode has a maximum limit of 15 actions to find the best association schema for the given topology. Because in the real environment such as the Nitos testbed at every iteration, e.g every episode, the Agent observes the environment almost one minute it is difficult and resource consuming to train the Agent over so many episodes.

Evaluation

Below we present some metrics during training process that are related with the efficiency of the DQN agent.

As we can see in figure 6.2, the total achieved throughput, i.e the sum of stations individual achieved throughput, increases over episodes and after 3000 episodes 170 Mbps is achieved. Assuming that theoretical maximum available throughput is set to 200 Mbps, 100 Mbps for each AP, we achieve a 85% utilization of the network.

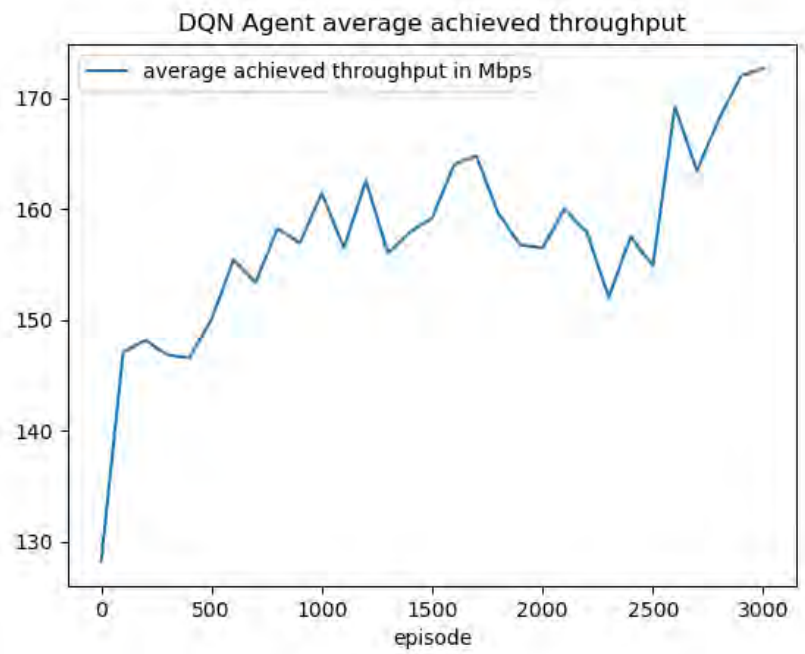

Figure 6.2: DQN Agent average achieved throughput

As we can see in figure 6.3, the average steps that the DQN Agent needs to find terminal state is is minimally reduced over episodes. It is important to note that a greater reduction would be desirable and this is something left as future work.

As we can see in figure 6.4, both minimum and average reward that DQN Agent per episode increase over episodes and after 3000 episodes the minimum reward is steadily positive and average reward about 700 is achieved.

The fact that both average total achieved throughput and average reward increase over episodes shows that the DQN agent training is well oriented and application in real environment has good prospects.

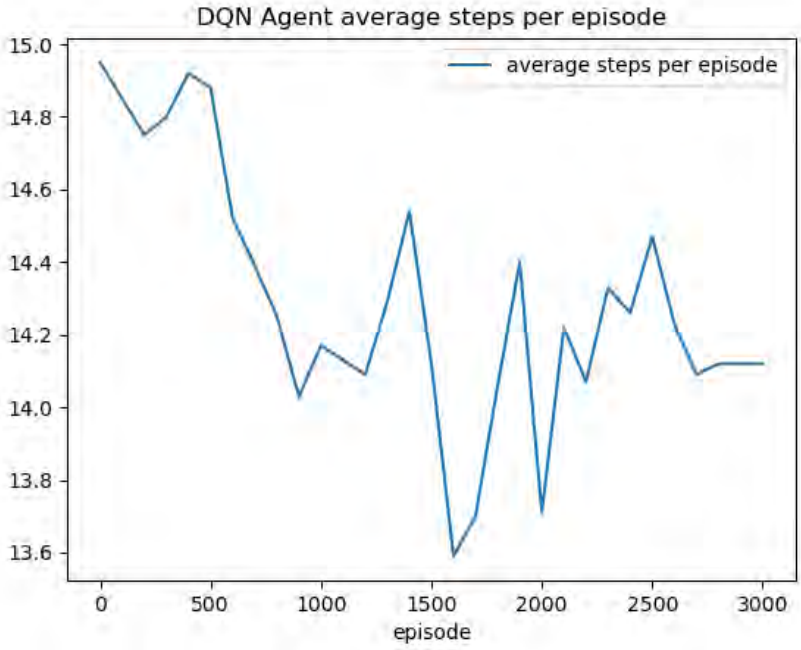

Figure 6.3: DQN Agent average steps per episode

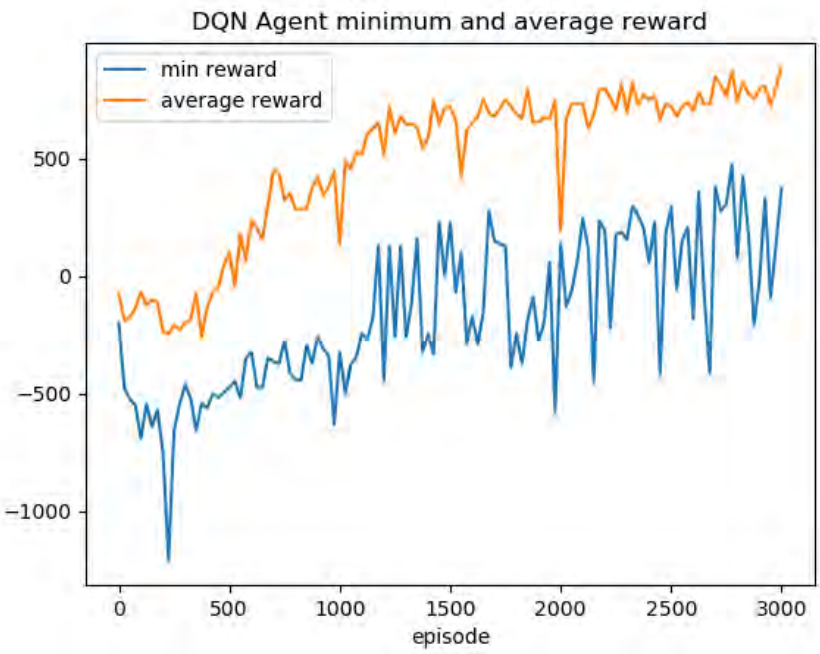

Figure 6.4: DQN Agent minimum and average reward

6.2.2 On-line training

Once the agent is connected to the real testbed topology the previously trained model is loaded and now at every iteration the agent explores the topology by requesting APs and updates its training data set with the current state. The DQN agent code is the same with the one presented in chapter 4. In order to evaluate the Agent's performance in the real environment we tested the DQN algorithm performance in different topologies compared to the default association method of 802.11ac protocol. At every experiment we connect each station at a random moment between 0 and 60 sec, we select a random throughput demand for each one and the Agent has also a maximum of 15 actions to find the best association schema. The network set up and the topology reset after each episode end is done automatically with shelh script commands and python3 scripts. Below we present a demo from an episode during run time. However, the evaluation of the DQN Agent in the real environment is left as future work because the training in the real testbed for an satisfying number of episodes is very time and resource consuming process.

Chapter 7

Conclusions

7.1 Summary and Conclusions

In the context of this thesis we presented the implementation of a mechanism for joint selection of central frequency, bandwidth and station association in WiFi networks. At first we had a brief historical analysis of the WiFi protocols up to the 802.11ac which is used in our implementation. Then we introduced a client steering mechanism which is used to achieve seamless and efficient handover of stations between access points. In our in our attempt to optimize station associations in a WiFi network with multiple stations and access points we suggest the use of Machine Learning and especially Reinforcement Learning. As we present in previous chapter we propose the DQN Agent in our approach for global management of the network. The training of the agent was initially done in a simulation environment and then the mechanism was implemented in the NITOS testbed (i.e a real environment).

During the first part of the training in the simulation environment the Agent seems to have responded satisfactorily. More specifically the evaluation of the Agent showed increase in total achieved throughput (i.e utilization of the network) and increase of the average reward over the pass of episodes. This encouraged us to test the application on a real network. However, training the algorithm in the testbed is extremely time and resource consuming proccess and so the training and evaluation in realistic topologies is left as future work.

7.2 Future Work

The most important prospect of the proposed mechanism is its application in real and large scale WiFi Networks, such as public places. As we mention above the DQN agent must be trained in the testbed environment where we can simulate a lot of realistic scenarios, such as topologies with high congestion and interference and different mobility scenarios. Further more, the DQN agent can be enriched with more metrics as input. Also optimizing neural networks internal hyper parameters can be examined more through experimental testing. Finally, as future work we aim to add in the Agent abilities channel selection for the access points among networks available frequencies. This process refers to the selection of central frequency for the access points and also the selection of the channel bandwidth among the 802.11ac available channel widths.

Bibliography

[1] Sibren De Bast, Rodolfo Torrea-Duran, Alessandro Chiumento, Sofie Pollin, and Haris Gacanin. Deep reinforcement learning for dynamic network slicing in ieee 802.11 networks. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, Paris, France, Sept. 2019.

[2] Piotr Gawłowicz and Anatolij Zubow. ns-3 meets openai gym: The playground for machine learning in networking research. In *MSWiM 2019*, Miami Beach, USA, Nov. 2019.

[3] M. Carrascosa and B. Bellalta. Decentralized ap selection using multi-armed bandits: Opportunistic ε -greedy with stickiness. In *2019 IEEE Symposium on Computers and Communications (ISCC)*, Barchelona, Spain, Jun. 2019.

[4] Mohamed Amine Kafi, Alexandre Mouradian, and Véronique Vèque. On-line client association scheme based on reinforcement learning for wlan networks. In *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, Marrakesh, Morocco, Apr. 2019.

[5] Eric Rozner, Yogita Mehta, Aditya Akella, and Lili Qiu. Traffic-aware channel assignment in enterprise wireless lans. In *2007 IEEE International Conference on Network Protocols*, Beijing, China, Oct. 2007.

[6] Chi-Yu Li, Chunyi Peng, Songwu Lu, Xinbing Wang, and Ranveer Chandra. Latency-aware rate adaptation in 802.11n home networks. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, Hong Kong, China, Apr. 2015.

[7] Keshav Sood, Shigang Liu, Shui Yu, and Yong Xiang. Dynamic access point association using software defined networking. In *2015 International Telecommunication Networks and Applications Conference (ITNAC)*, Sydney, NSW, Australia, Nov. 2015.

[8] Wangkit Wong, Avishek Thakur, and S.-H. Gary Chan. An approximation algorithm for ap association under user migration cost constraint. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, San Francisco, CA, USA, Apr. 2016.

[9] Ouldooz Baghban Karimi, Jiangchuan Liu, and Jennifer Rexford. Optimal collaborative access point association in wireless networks. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, Toronto, ON, Canada, Apr. 2014.

[10] Nitlab. `https://nitlab.inf.uth.gr/NITlab/`. Accessed: 29-06-2023.

[11] Nitlab outdoor testbed. `https://nitlab.inf.uth.gr/NITlab/outdoor-hidden`. Accessed: 29-06-2023.

[12] Nitlab intdoor testbed. `https://nitlab.inf.uth.gr/NITlab/indoor-hidden`. Accessed: 29-06-2023.

[13] Ath10k. `https://github.com/kvalo/ath10k-firmware`. Accessed: 29-06-2023.

[14] Wireshark. `https://www.wireshark.org/`. Accessed: 29-06-2023.

[15] iperf. `https://iperf.fr/`. Accessed: 29-06-2023.

[16] Tensorflow. `https://www.tensorflow.org/`. Accessed: 29-06-2023.

[17] Pandas dataframe. `https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html`. Accessed: 29-06-2023.

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]